Package Managers are Great, Let’s Make them Greater

An article by Space Banana

Introduction

Package management is a software distribution method that relies on a package manager, the component that functions on the client, to grab a desired piece of software from one or multiple repositories hosted on a different machine. Package managers are very common in programming languages, for downloading libraries, but they are also the standard method of software installation in Unix-like operating systems, such as Linux-based operating systems, Haiku, and systems that originate from the BSD roots.
Package management is great, but it could be better. In this article, I will talk about what is wrong with modern package management and how it could be better. I will talk about package mangement in the context of operating systems and software distribution, and not library package management for programmers.

Too much binary-based package management

A binary-based package management system can only fetch software in compiled form. This is the most common type of package management used in operating systems. We have systems like Debian, Arch Linux, Fedora and many others that use binary package management systems.

Many times, we would have a better experience with a source-based packaging system. With a source-based package manager, we can fetch the source code of the software and let the package manager build it from source. This allows us to increase the diversity of CPU architectures with little effort, while also letting us specify custom build flags, compile options and dependency selection. You might not desire any of these abilities of a source-based package manager, but the ultimate advantage is one that helps all users, even those who only install pre-compiled software: source-based package managers provide the package compilation procedure for the user, which increases transparency and makes it easier for volunteers to package software.

Volunteer packaging

Most system package repositories contain software that was packaged by a small team of official maintainers. This makes it harder to maintain large repositories, as the maintenance and packaging teams are small. By introducing a transparent and public methodology for packaging software for your system, mostly common in source-based package managers, volunteers have an easier time to contribute.

Can the security of your system be compromised by allowing volunteers to publish packages? Yes, if not done carefully, but we have examples of how it can be done carefully. NixOS and FreeBSD allow volunteers to package software and maintain said packages. The user must first make a request to upload this package, and the request has to be verified by an official member of the system maintenance team. If the package is approved, the user becomes a maintainer for said package.

With this methodology, the package repository can be significantly larger, with the help of many more volunteers. A large repository, such as NixOS's, can be used as a general-purpose repository with the capability of replacing other methods of getting your software.

Package release cycles

Operating system package repositories have 2 major types of release methods: point release and rolling release.

A rolling release system updates packages as they and their respective testing are ready. The repository doesn't have a version by itself, and each package updates independently. A point release system, however, updates packages by major version. Packages that are updated and tested usually must wait until they come in the next major update of the repository. There are exceptions, such as the package update bringing a small patch or a security fix, which in that case the package will be updated swiftly. In some cases, such as Fedora's packaging system, software is updated in-between major repository versions, unless the update is significantly large, or the package is crucial to Fedora's stability.

People's preferences on the type of update system are subjective. A point release brings more system stability and predictability, creating a system that is more reliable for serious work. A rolling release brings the latest and greatest versions of software.

Having the whole system controlled by a rolling repository is very dangerous, as the system is at risk of instability and unpredictable bugs, but having the whole system controlled by a point release repository slows down the updates of our everyday software that we would have installed from their official website otherwise. Some systems, such as Debian, have stable release systems that are very slow. In case of Debian, the stable repository normally updates twice a year, which is not ideal for our everyday software. Debian does have faster-updating branches of this repository but they are not as stable, of course. Other systems, such as Arch Linux, have the whole system controlled by a rolling repository, which makes Arch occasionally push unstable versions of software, even if said software is critical to the system, like the bootloader!

We do not need to adopt extreme positions, we can embrace a middle-ground for this problem. We can maintain repositories that update twice or four times a

year, which are reasonable update frequencies. NixOS's stable repository updates twice a year, but even then, for daily use software that's a bit slow, updating four times a year would be more appropriate.

Instead of adopting a middle-ground, we can also mix both concepts. We can inspire ourselves in the packaging system of FreeBSD and maintain a stable, minimal core for the system that updates twice a year, while separating the rest of the software in a repository that updates four times a year or in rolling release fashion. This is the perfect model for using native system packaging for both stable and secure system maintenance and for downloading our everyday programs.

Mixing native packages with third-party

A common practice in the Linux world is for users to use both their native system package manager, as well as a third-party package manager, like Flatpak and Snap. This approach allows the user to not fully have to rely on the native package repository, making the repository suitable even if it lacks software or updates slowly, and instead install their everyday software from third-party packaging systems.

This approach is very good in theory, but in practice we lack an ideal package manager for this, as the current ones have their set of issues. Snaps have many issues and the software is closed source, making the package manager and repository undesirable. Flatpak is not well designed for CLI and TUI software, and so the Flathub repository also lacks most of them if not all, and does not share dependencies between packages, bloating the user's disk, as well as other issues. Both Flatpak and Snap are also Linux-only. Thankfully, we can use NixOS's Nix package manager outside NixOS.

The Nix package manager can be used on any other Linux-based system, as well as MacOS, and FreeBSD support is experimental and (slowly) on the works. Nix, with its giant repository, is very suitable for the role of a third-party package manager. Dependencies are also shared between packages, and the package manager does not make it less than ideal for you to install and run CLI and TUI software.

Dependency hell

Dependency hell is a threat to most package managers. Dependency hell is defined by an installation or update of a package which creates conflicts between dependencies or their versions, missing dependencies, or other common library and dependency issues, at the risk of breaking your system or the package manager. It's very hard to avoid dependency hell, and for most package managers not much can be done other than to rely on the maintainers' responsibility to provide stable updates with enough quality control. NixOS is a rare case as it is immune to dependency hell, due to its

declarative packaging system. Flatpak is also immune to dependency hell, but mostly because dependencies are duplicated, since each package comes with what it requires regardless of what is already installed in the system.

The user should never be responsible to fix system breakage caused by dependency hell or the lack of quality control from the maintainer side.

User experience

The user experience of using a package manager is also important. A powerful but sane package manager will have a pleasant interface and won't create much friction or problems when you have to update your system, change repositories, manage packages, and perform other functionality.

Arch Linux's package manager, pacman, has a great CLI, providing a great user experience when it comes to managing packages. FreeBSD's package manager, on the other hand, has its own issues with user experience. Upgrading the core system is a painful and very manual task, that can cause system breakage, causing a negative impact on user experience.

Upstream packages and security

Operating system package repositories contain downstream distributions of software, which means the software is packaged by someone other than the official developers of the software. This is a must, since upstream developers do not have the resources or time to package their software for every single packaging system.

In case of third-party package repositories, like the repositories for Flatpak and Snap, many programs are packaged by the official developers, which ensures the package is up-to-date and works as intended, but we have to entrust each developer the safety of our system. In system package repositories, we instead entrust our operating system maintainers, resulting in a much tighter security.

While it's great that developers publish their software to these repositories, it's always great if they also publish their software in unpackaged form, usually as binaries, for those who do not want to rely on these package systems.

Conclusion

Installing software from a package manager and its respective repository is a great way to intsall and manage software and its dependencies, especially if said package manager is your operating system's own package manager. While package managers are great, we can vastly improve their experience by raising quality control, to avoid dependency hell. The user experience and how

pleasant and reliable it is to manage packages and updates should also stay in consideration. System stability is crucial, but software update frequency is also important for our everyday programs, and so using a point release repository for the core system and a rolling or quarterly repository for all other software is a great solution to this problem. CPU architectures should not be left behind, we should adopt more the use of source-based package management, and said packaging system also provides the packaging methodology to the user, making it more transparent for people to package software themselves. If volunteers can package software safely, then the repository can grow significantly in size, lowering the need for third-party package managers and repository. As a last resource, using a third-party package manager is a great solution too.